**人工智能原理实验**

**实验二、启发式搜索算法求解8/15数码问题**

**组长：**凌翔鸿（学号3119000854）

**组员：**林方泽（学号3119000980）、邱俊文（学号3119001159）、莫嘉楹（学号3219001092）、吴泽鑫（学号3119001082）

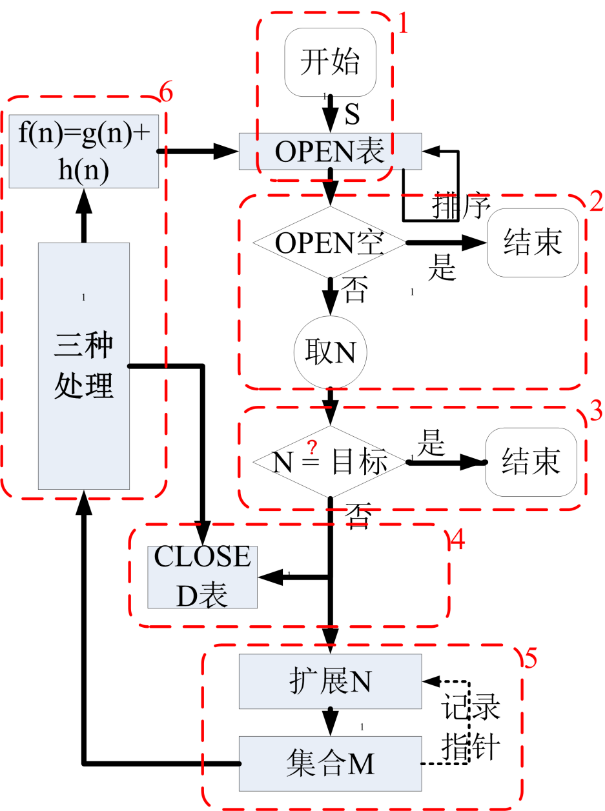
**指导老师：**高怀恩

**一、实验目的：**

熟悉和掌握启发式搜索的定义，了解启发书搜索的估价函数（f=g+h）和算法过程，并利用启发式算法求解八数码难题，理解求解流程和搜索顺序。

**二、实验内容：**

（1）本实验使用了A\*算法，在满足h（n）<=实际代价的前提下，h（n）越大，则它的搜索状态越少，算法越高效，其流程图如下：



（2）估价函数修改如下：

function cost=calH0(node,des)

cost = 0;

end

function cost=calH1(node,des)

cost = length(find(node.con~=des));

end

function cost=calH2(node,des) % 各数码移到目的位置所需移动的距离的总和

cost = 0;

for i = 0:(numel(node.con)-1)

[x0, y0] = find(des == i);

[x, y] = find(node.con == i);

cost = cost + abs(x - x0) + abs(y - y0);

end

end

function cost=calH3(node,des) % 对每一对逆转数码乘以一个倍数

cost = nixudui(node.con) \* 3;

end

function cost=calH4(node,des) % 将位置不符数码数目的总和与3倍数码逆转数目相加

cost = (nixudui(node.con) \* 3 + length(find(node.con~=des)));

end

function cost=calH5(node,des) % 各数码移到目的位置所需移动的距离与逆转数码数目相加

cost = nixudui(node.con);

for i = 0:(numel(node.con)-1)

[x0, y0] = find(des == i);

[x, y] = find(node.con == i);

cost = cost + abs(x - x0) + abs(y - y0);

end

end

（3）实验结果（‘\’代表代码跑不出来）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **表格一** | **启发式函数 h(n)** | | | | |
|  | **宽度优先算法（H0）** | **不在位数**  **（H1）** | **（H2）** | **（H3）** | **（H4）** |
| **初始状态** | **初始状态 目标状态**  initMatrix (:)=**[8 1 0 5 7 2 6 3 4] [0 1 2 3 4 5 6 7 8]** | | | | |
| **目标状态** |
| **最优解**  **(移动步数)** | **\** | **25** | **25** | **27** | **29** |
| **扩展节点数**  **(close表）** | **\** | **13898** | **2999** | **1094** | **421** |
| **生成节点数**  **(open表）** | **\** | **6906** | **1702** | **681** | **261** |
| **运行时间（秒）** | **\** | **340.066** | **22.714** | **4.769** | **0.731** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **表格二** | **启发式函数 h(n)** | | | | |
|  | **宽度优先算法（H0）** | **不在位数**  **（H1）** | **（H2）** | **（H3）** | **（H4）** |
| **初始状态** | **初始状态 目标状态**  initMatrix (:)=**[1 0 4 3 6 7 2 8 5] [0 1 2 3 4 5 6 7 8]**  **C:\Users\gao\AppData\Local\Temp\1602987782.png 02** | | | | |
| **目标状态** |
| **最优解**  **(移动步数)** | **\** | **24** | **24** | **24** | **26** |
| **扩展节点数**  **(close表）** | **\** | **9309** | **1296** | **815** | **1028** |
| **生成节点数**  **(open表）** | **\** | **4779** | **750** | **526** | **654** |
| **运行时间（秒）** | **\** | **164.432** | **5.865** | **2.393** | **3.754** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **表格三** | **启发式函数 h(n)** | | | | |
|  | **宽度优先算法（H0）** | **不在位数**  **（H1）** | **（H2）** | **（H3）** | **（H4）** |
| **初始状态** | **初始状态 目标状态**  **initMatrix** (:)**= [7 4 9 6 3 12 14 10 2 5 0 15 11 13 1 8] [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]** | | | | |
| **目标状态** |
| **最优解**  **(移动步数)** | **\** | **\** | **\** | **\** | **\** |
| **扩展节点数**  **(close表）** | **\** | **\** | **\** | **\** | **\** |
| **生成节点数**  **(open表）** | **\** | **\** | **\** | **\** | **\** |
| **运行时间（秒）** | **\** | **\** | **\** | **\** | **\** |

（跑了超过半天实在跑不出来......）

**三、实验总结：**

（1）心得

通过本次实验，我们对启发式搜索有更近一步的认识，估价函数不同，算法效率也会有所不同，但普遍都比广度优先搜索和深度优先搜索要快许多（一般情况下）。

估价函数定义不当，可能会导致搜索出的结果为局部最优，有时候虽然是局部最优，但其搜索速度大大提升。

由上面三个表格可以看出，结果步数越多，搜索次数可能是以指数级增长的，某个算法在某种情况很快，不代表所有结果都很快。

总之，完成本次实验后，我们把理论上的东西落地到代码上实现，提高了我们的编程能力，更加熟练的掌握了启发式搜索，并且理解到启发式搜索的优点所在。

（2）改进

由于在满足h（n）<=实际代价的前提下，h（n）越大，则它的搜索状态越少，算法越高效，我们改进了估价函数，即calH5估价函数如下：

function cost=calH5(node,des) % 各数码移到目的位置所需移动的距离与逆转数码数目相加

cost = nixudui(node.con);

for i = 0:(numel(node.con)-1)

[x0, y0] = find(des == i);

[x, y] = find(node.con == i);

cost = cost + abs(x - x0) + abs(y - y0);

end

end

在表一的情况中，calH5所用时间与calH4相近，但搜索出来的步数要比calH4更优；在表二的情况中，calH5花时约0.4s，要比calH0~4的算法要快许多，搜出的结果为最优，可见我们自定义的calH5函数要比calH0~4函数要更高效。

除此之外，我们还把启发式搜索在python上实现了，具体代码见附带文件，该代码能够完成calH0~4的启发式搜索。